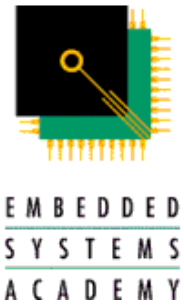
Philips P89LPC932 Flash Microcontroller In-Application Programming Library Manual

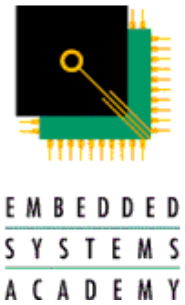
Version 1.00 (24-Feb-2003), by AA

Contents

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



1. Introduction

The P89LPC932 8-bit Microcontroller from Philips provides a BootROM which contains a routine to allow an application to perform programming operations on the internal Flash memory. The routine may be called with varying values in registers R3 – R7 and the Accumulator to perform the following operations:

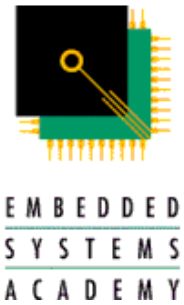
- Read and program bytes in the Flash memory
- Read the IAP version
- Read and write the User Configuration Byte
- Read and write the Boot Vector and Status Byte
- Read and write the Security Bytes
- Erase sectors and pages
- Read sector and global CRC values

The LPC932 IAP Library (LPC932IAPLIB) provides a means to perform all the operations by making function calls in C. No assembler needs to be written.

The library also provides pre-defined values to ease the use of certain operations.

The first part of this manual describes each function in turn. The function operation, parameters passed and parameters returned are detailed.

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



2. Using the Library

2.1 Adding to a Project

The library consists of a source file:

```
LPC932IAPLIB.A51
```

and a C header file:

```
LPC932IAPLIB.H
```

The library is compatible with both Raisonance and Keil 8051 Compilers. It has been written and tested with the following versions:

- Keil C51 v7.03
- Raisonance RC51 v03.03.28

To use the library simply add the library source file to your project and include the header file in all C source files that use the library.

2.2 Calling Functions

Before any calls to the library can be made, a call to `iap_init` must be made. Please refer to the function description in this manual for details.

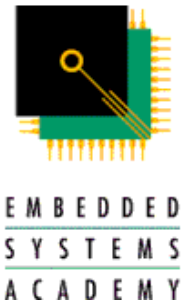
Each function returns the execution result in the global varible iap_status, which is defined as an unsigned char. iap_status may contain either IAP_OK which means the operation was successful or one or more of the following error values:

IAP_INTERRUPTED	- operation was interrupted by an interrupt
IAP_SECURITY	- security violation occurred
IAP_ERROR	- miscellaneous error occurred
IAP_ABORT	- operation was aborted.

The library may be configured to feed the watchdog immediately before performing an IAP operation. This should be used if the watchdog is enabled and the timeout period for the watchdog is longer than 2ms. If the timeout period is shorter than 2ms then IAP operations will not be able to complete in time, even with a feed immediately before the operation. In this situation the watchdog must be disabled before making calls to the IAP library.

The IAP library may be configured to disable interrupts for the duration of IAP operations. If interrupts are not disabled then there exists a possibility that an IAP operation may be interrupted. If that occurs then the function will return with iap_status set to IAP_INTERRUPTED. A check must be made for this and the function may be called again if the operation should be reattempted. Note however, that if any interrupt occurs more frequently than 2ms, then some IAP

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



operations may never be completed with interrupts enabled. In that situation the library may be configured to disable interrupts or specific interrupts may be disabled before calling functions in the library.

2.3 Execution Environment

The functions are registerbank independent and do not use absolute register addressing.

Before the IAP functionality of the LPC932 may be used, the brown-out detection feature must be enabled by setting the BOE bit in User Configuration Byte 1 (UCFG1). Although the possibility exists of writing to UCFG1 via an IAP routine, it is strongly recommended that the brown-out detection is NOT enabled via an IAP call, but is instead enabled via parallel programming or In-System Programming (ISP).

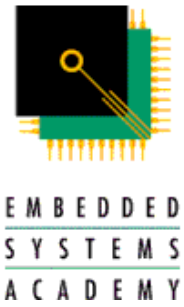
The library upon initialization completes the enabling of the brown-out power down by clearing bits 0, 1 and 5 in PCON.

Failure to enable the brown-out detect may result in IAP operations either executing incorrectly or aborting (and therefore setting iap_status to IAP_ABORT).

2.3 Locating Library Functions

When calling the erase functions, the IAP functions themselves and the library code used by the IAP library must not be located inside the sector or page being erased.

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3. Library Functions

3.1 iap_init

Description

Configures the IAP library. This function must be called before any other functions in the library.

Prototype

```
void iap_init(unsigned char configuration);
```

Arguments Passed

Configuration may be either be the value zero, or one or more of the following options:

IAP_WATCHDOGFEED	- configures the library to feed the watchdog immediately before performing an IAP operation
IAP_INTDISABLE	- configures the library to disable all interrupts while performing IAP operations.

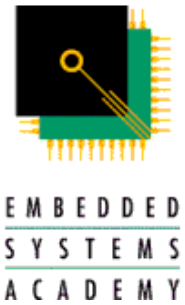
Value Returned

None

Example

```
iap_init(IAP_WATCHDOGFEED);
iap_init(IAP_WATCHDOGFEED | IAP_INTDISABLE);
iap_init(0);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.2 iap_read_version

Description

Reads the version of the IAP code programmed onto the device at the factory.

Prototype

```
unsigned char iap_read_version(void);
```

Arguments Passed

None.

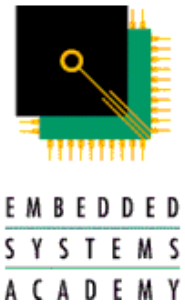
Value Returned

The IAP version number.

Example

```
version = iap_read_version();
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.3 iap_write_code

Description

Programs code memory from a buffer. The start address may be anywhere in code memory, however the bytes programmed must fit entirely within a 64 byte page. Therefore the largest buffer size supported is 64 bytes. The buffer must be located in DATA memory and therefore directly addressable.

Prototype

```
void iap_write_code(unsigned char data *pbuffer,
unsigned int address, unsigned char length);
```

Arguments Passed

Pbuffer is a pointer to the buffer containing the data to program
Address is the start address to program the data at in code memory
Length is the number of bytes in the buffer to program (1 – 64).

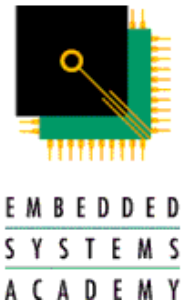
Value Returned

None.

Example

```
unsigned char data buf[4];
buf[0] = 0xAA;
iap_write_code(buf, 0x1402, 4);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.4 iap_erase

Description

Erases a sector or page in code memory.

Prototype

```
void iap_erase(unsigned char type,
unsigned int address);
```

Arguments Passed

Type indicates whether to erase a sector or page and is one of the following values:

IAP_SECTOR
IAP_PAGE

Address is the starting address of the sector or page to be erased.

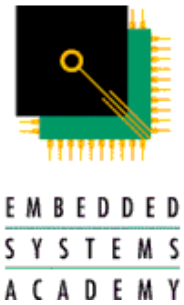
Value Returned

None.

Example

```
iap_erase(IAP_PAGE, 0x1440);
iap_erase(IAP_SECTOR, 0x1400);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.5 iap_read_sector_crc

Description

Reads the 32-bit CRC value for a sector of code memory.

Prototype

```
unsigned long iap_read_sector_crc(unsigned int
address);
```

Arguments Passed

Address is the starting address of the sector whose CRC should be read.

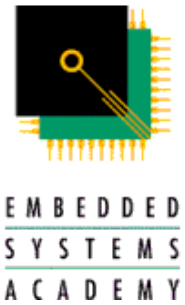
Value Returned

The 32-bit CRC.

Example

```
unsigned long crc;
crc = iap_read_sector_crc(0x1400);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.6 iap_read_global_crc

Description

Reads the 32-bit CRC for the entire code memory.

Prototype

```
unsigned long iap_read_global_crc(void);
```

Arguments Passed

None.

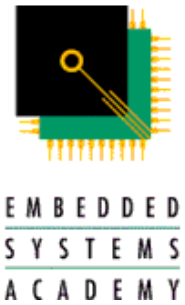
Value Returned

The 32-bit CRC.

Example

```
unsigned long crc;
crc = iap_read_global_crc();
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.7 iap_read_code

Description

Reads a location in code memory.

Prototype

```
unsigned char iap_read_code(unsigned int address);
```

Arguments Passed

Address is the address of the location to read.

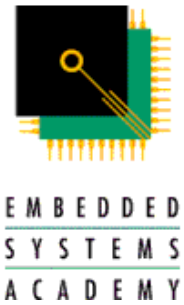
Value Returned

The value stored in the location.

Example

```
unsigned char byte;
byte = iap_read_code(0x1401);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.8 iap_read

Description

Reads from a configuration byte.

Prototype

```
unsigned char iap_read(unsigned char name);
```

Arguments Passed

Name is the name of the byte to read, and is one of the following:

IAP_UCFG1
IAP_BOOTVECTOR
IAP_STATUSBYTE
IAP_SECURITYBYTE0
IAP_SECURITYBYTE1
IAP_SECURITYBYTE2
IAP_SECURITYBYTE3
IAP_SECURITYBYTE4
IAP_SECURITYBYTE5
IAP_SECURITYBYTE6
IAP_SECURITYBYTE7

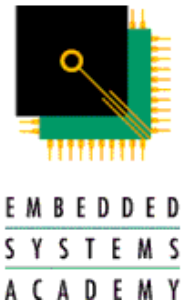
Value Returned

The value of the configuration byte.

Example

```
unsigned char statusbyte;
statusbyte = iap_read(IAP_STATUSBYTE);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



3.9 iap_write

Description

Writes to a configuration byte.

Prototype

```
void iap_write(unsigned char name,
unsigned char value);
```

Arguments Passed

Name is the name of the configuration byte to write to and is one of the following:

IAP_UCFG1
IAP_BOOTVECTOR
IAP_STATUSBYTE
IAP_SECURITYBYTE0
IAP_SECURITYBYTE1
IAP_SECURITYBYTE2
IAP_SECURITYBYTE3
IAP_SECURITYBYTE4
IAP_SECURITYBYTE5
IAP_SECURITYBYTE6
IAP_SECURITYBYTE7

Value is the value to write to the configuration byte.

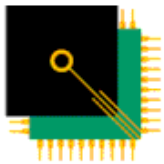
Value Returned

None.

Example

```
iap_write(IAP_UCFG1, 0x23);
```

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com



EMBEDDED SYSTEMS ACADEMY

4. Examples

The example included with this package demonstrates some of the features of the library. Included is a project file for the Keil Compiler (uVision2) and a project file for the Raisonance Compiler (RIDE). Both Compilers use the same source files.

KLPC932IAPLIBEXAMPLE.UV2	- Keil uVision2 Project file
KLPC932IAPLIBEXAMPLE.HEX	- Keil generated HEX file
KLPC932IAPLIBEXAMPLE	- Keil generated AOF file
KLPC932IAPLIBEXAMPLE.M51	- Keil generated Map file

RLPC932IAPLIBEXAMPLE.PRJ	- Raisonance RIDE Project file
RLPC932IAPLIBEXAMPLE.HEX	- Raisonance generated HEX file
RLPC932IAPLIBEXAMPLE	- Raisonance generated AOF file
RLPC932IAPLIBEXAMPLE.M51	- Raisonance generated Map file

MAIN.C	- Example main source file
LPC932IAPLIB.A51	- IAP Library source file
LPC932IAPLIB.H	- IAP Library header file
OSC.C	- Routines to configure the oscillator
UART.C	- Routines to configure and use the UART.

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com

5. Library Run Time Requirements

5.2 Total Code Size

The entire library occupies 182 bytes of CODE memory.

5.3 Data Size

The library requires one byte and two bits of DATA memory.

5.4 Stack

Each function performs the usual pushing of the return address on the stack for a function call. An additional return address is pushed onto the stack during most of the functions in the library.

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com
info@esacademy.com